LLNL-TR-592878

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

ASC Co-design Proxy App Strategy

J. R. Neely, M. Heroux, S. Swaminarayan

October 19, 2012

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

ASC Co-design Proxy App Strategy

Mike Heroux *(maherou@sandia.gov)*
Rob Neely *(neely4@llnl.gov)*
Sriram Swaminarayan *(sriram@lanl.gov)*

Introduction

The NNSA labs representing the ASC program are coordinating on the development and release of a portfolio of proxy applications to be used in co-design. While each lab will develop their own set of proxy apps that represent the applications and algorithms of interest to them, we propose a common set of criteria to describe and measure proxy apps, and a strategy aimed at minimizing duplication and maximizing impact across the entire ASC proxy app portfolio.

Proxy application (or proxy apps for short) is a catchall term for the simplification of characteristics of real applications that are of interest to DOE. These proxy apps are used in the co-design process as concrete examples for component and system designers to understand our software requirements. The proxy apps are generally openly available to the co-design community and, unlike benchmarks, are intended to facilitate the *two-way* communication required by co-design to optimally evaluate trade-offs in the system hardware, and inform the developer community of programming best practices for emerging architectures.

Proxy apps capture a subset of the characteristics typical in real applications, such as:

- Algorithms
- Data structures and memory layouts
- Parallelism and I/O models
- Languages and coding styles

To enable a rapid exploration and evaluation of the above application characteristics for future architectures, the proxy applications are designed to be a much smaller code base and simpler to understand than the full applications. However, this leads to a natural tension between proxy app code simplicity and accurately representing the full application of interest. As a code developer develops one or more proxy apps, some application characteristics will be emphasized and others deemphasized. Likewise, the value of proxy applications lies in the entire portfolio, not in any single proxy application. No single application (proxy or otherwise) could possibly represent all of the interests of DOE, so we are likewise presented with the challenge of optimizing the overall portfolio to contain as few as possible (simplicity) with maximum coverage (accuracy).

In addition to being used for co-design of system software & hardware, proxy apps are also used for the exploration of new programming models and algorithms, and as a low overhead way for developers to try new ideas in a smaller code base. These proxy apps typically strive to isolate a particular characteristic or algorithm over being representative of a full application.

Standardized Taxonomy for Proxy Apps

The ASC program is standardizing on the terminology below to differentiate different types of proxy apps, in order from simplest to most complex:

LLNL-TR-592878

Kernels: These are one or more small code fragments or data layouts that are used extensively by the applications and are deemed essential to perform optimally on next generation advanced systems. These are useful for testing programming methods and performance at the node level, and typically do not involve network communication (MPI). Their small size also makes them ideal for doing early evaluation and explorations on hardware emulators and simulators.

Skeleton apps: These apps reproduce the memory or communication patterns of a physics application or package, and make little or no attempt to investigate numerical performance. They are useful for targeted investigations such as network performance characteristics at large scale, memory access patterns, thread overheads, bus transfer overheads, system software requirements, I/O patterns, and new programming models. Skeleton also may allow the release of more applications as non-export controlled by removing mathematical or algorithmic details while still conveying useful performance information.

Mini apps: These apps combine some of the dominant numerical kernels (or subsets thereof) contained in an actual stand-alone application and produce simplifications of physical phenomena. This category may also include libraries wrapped in a test driver providing representative inputs. They may also be hard-coded to solve a particular test case so as to simplify the need for parsing input files and mesh descriptions.

Compact apps: These apps are the most representative of actual applications, and likewise usually the largest and most complex. In some cases compact apps may be full-fledged physics packages (with drivers), and as such may be more restricted in their distribution.

In addition to the above categories that define proxy apps at varying levels of size and complexity, we may extend and/or combine any of the above descriptions with the term *coupled* to describe proxy apps that combine two or more proxies from different physical domains, e.g. *coupled mini app*. This will assist in exploring unique aspects of multi-physics ASC Integrated Codes. For example:

- Ensuring code optimizations or hardware tradeoffs that help one package (or proxy) don't negatively impact the other
- Novel parallelization models, such as running different physics packages (or operators) on different sets of nodes
- Impacts of additional data motion required by mapping of one mesh topology onto another

Documentation Template

It is an important responsibility of the DOE co-design efforts to present an understandable and systematic description of our proxy apps to the co-design community. While the source code itself is the ultimate documentation, it is important that we also attempt to standardize on the descriptions of our proxy app portfolio. This will allow us to ensure a consistent level of high level of documentation across ASC co-design proxies, and support app-to-app comparisons.

Some characteristics can be captured very succinctly and are best captured in a high level table allowing at-a-glance comparison of the apps in the portfolio. These include:

Summary Description of Proxy

- Name of the proxy app
- Short description of the domain, or the full application it is a proxy for

- Type of proxy (kernel, skeleton, … see above)
- Home institution or co-design center
- Language(s) used
- Programming models used (all that apply), e.g.
    - MPI
    - OpenMP
    - OpenACC / CUDA
    - PGAS
    - SIMD/vectorization
    - Other
- External library dependencies
- Lines of code (not including blanks and comments)
- Release restrictions, e.g.
    - Open source
    - Restricted export-control (EAR99)
    - Restricted DOE
    - Restricted lab
- Version number
- Point of contact
- Location of open source repository

Other characteristics require a bit more description, and are not appropriate for a high-level table such as the previous set of data. These descriptions should appear in a more detailed document that is openly published.

Detailed Technical Description of Proxy

The top-level bullets in **bold** are suggested section headers. Text and bullets in italics are suggested topics to

- **Introduction**.
- **Requirements** (use problem domain language in this section):
    - *Why did you create this proxy?*
    - *What is the perceived weakness of current approaches?*
    - *What opportunities are possible?*
    - *What benefits can be realized by success?*
- **Analysis** (use computing domain language in this section):
    - *What is the proxy app intended to explore? E.g.*
        - *Single processor performance.*
        - *Memory systems*
        - *Large system scaling (e.g. MPI)*
        - *Fine-grained concurrency (threading)*
        - *Load balancing*
        - *Alternate programming models*
        - *Other (specify)*
- **Design** (describe the proxy including analytic and discrete models)
    - *How is this proxy representative of the full application, and under which conditions? What was simplified that might affect its accuracy at illustrating impacts of co-design?*
    - *What are the restrictions on modifications one should be wary of? E.g.*
        - *Numerical accuracy*

        - *Data structures*
- **Implementation** (describe reference implementation and any derivatives)
    - *What are the major software components and functions?*
    - *Which components/functions are of most interest for co-design activities?*
    - *Platforms that it's been optimized for (e.g. accelerators, torus interconnect, SIMD)*
- **Physical models**
    - *Describe the underlying physics and/or equations. Ideally, this would be in enough detail to allow someone to reproduce the proxy app (see LULESH for an example)*
- **Inputs and outputs**
    - *What are the use cases for the sample test problem?*
    - *What inputs are required for various test cases?*
    - *What outputs are generated?*
    - *What are the sample performance results obtained by DoE on known hardware?*
- **Testing** (discuss verification and validation topics)
    - *How can you tell if it ran correctly?*
    - *What validation (proxy vs full app) comparisons are available?*
- **Future Plans/Retirement** (describe what should be done in the future)
    - *What omitted functionality could be added in the future?*
    - *Which alternate methods or algorithms that have been explored or need exploring?*
    - *What criteria should be considered for retiring this proxy?*
- **Deployment** (Describe availability, distribution, etc.)
    - *Point of contact info / author*
    - *Dependencies on other software*
    - *Build and run instructions*
    - *Download process (web site, contact email, etc…)*
    - *Contribution process (repository access)*
    - *Bibliography entry for references*
    - *List of reference sources (e.g. talks and papers using the proxy)*

Standard MS-word and LaTeX templates will be made available to facilitate documentation by proxy app authors.

Metrics

In addition to standardized documentation criteria, we also propose to approach the evaluation of the software metrics with a measure of rigor. These metrics are a first step toward being able to answer the following questions:

1) How representative are these proxies of their full applications?
2) How do the proxy apps compare to each other across the portfolio?
3) How do updates to the proxy app affect key characteristics, besides raw performance on current architectures?

For example, it is one thing to state that an application is "memory bound", but quite another if you can back that up by quantifying memory accesses per flops, cache hit rates, etc. The following is list of suggested metrics that are easily collected using widely available tools.

- Instruction mix (e.g. floating point, integer, branch, memory)

- Computational intensity (flops per memory access)
- Percentage of floating point ops that are vectorized (SIMD)
- Range of instructions/cycles per loop iteration
- L1/L2/TLB hit rates
- Amount of serial sections
- Parallel efficiency at scale (specify platform + MPI tasks)
- Working set size (high/low water mark)
- Message statistics (message sizes, frequency)
- I/O statistics (amount read/written, I/O model – collective, single proc, file-per-proc)

It is understood that many of these metrics will be dependent on the hardware and compilers used for a given application, and thus the platform, compiler, and other relevant information should also be captured for each set of metric data.

Source & Data Management

At first glance, managing proxy application source code and generated data may seem straightforward. However, there are two major challenges that must be addressed, especially as a proxy becomes heavily used.

- **Version control for refactored variants:** One major activity when working with proxies is refactoring the reference code in order to explore different computation and data organization strategies, programming models and algorithms. These activities will potentially generate many variations of the reference proxy and the proxy owner will often want to capture these. Policies and strategies for managing these versions will be documented. One approach to this issue is to make the reference version support MPI and OpenMP, either, both or neither. Other variations are then kept independently in the repository.

- **Data generation, management and analysis:** When executing a proxy application and generating performance data, we often have studies that generate hundreds of samples. Collecting and analyzing this data is best done with database and post-processing tools. Common data formatting and post-processing strategies would be useful. One approach to this issue is to use YAML for data formatting. YAML is a human-readable ASCII-based format that also supports interactions with XML and SQL. It has been used with some success on existing proxy applications.

The ASC co-design projects at each lab will work together to share best practices, with a goal of using common procedures when possible.